Devoir II

8INF840 – Structures de données avancées et leurs algorithmes

Été 2018

Aymen Sioud

UNIVERSITÉ DU QUÉBEC À CHICOUTIMI
Département d'Informatique et de Mathématique

Consignes

- **A remettre au plus tard le 22 juin à 18h (heure de l'est).**
- **Vous devrez utilisez Visual Studio 2017.**
- **Le travail doit être effectué en C++.**
- **Il ne devra pas y avoir de dossier Debug**
- **Le travail devra être remis via la plateforme moodle en un seul répertoire compressé.**
- **Le travail remis devra être nommé :**

  Devoir2_nom1_nom2_nom3_nom4_8INF840_E2018.
- **Vous devrez remettre en plus de vos projets Visual Studio, un rapport contenant les captures d'écrans et les traces de vos exécutions et de vos différentes fonctionnalités pour chacun des exercices.**

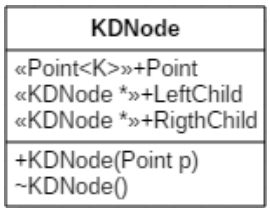
Bon travail !

Exercice 1 : Kd-Tree

Les Kd-Tree est une structure de données qui permet de gérer le partitionnement de l'espace autour de valeurs et permet l'utilisation de nombreux algorithmes de recherche très rapide répondant à différent problèmes, tels la recherche du plus proche voisin ou la recherche dans une plage de valeur.

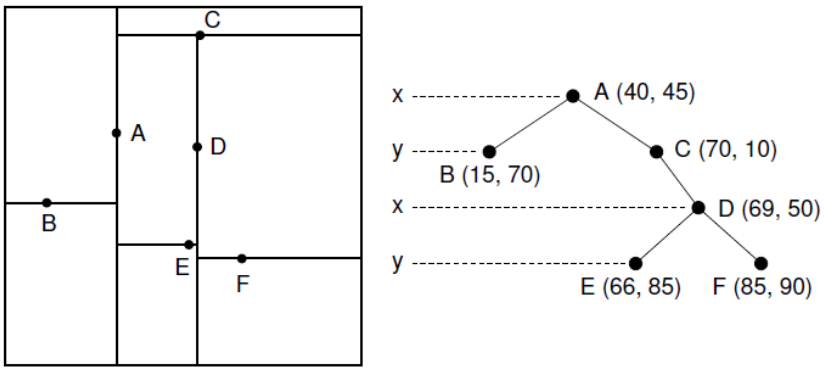
Les arbres k-d sont des arbres binaires, dans lesquels chaque nœud contient un point en dimension k. Chaque nœud non terminal divise l'espace en deux demi-espaces. Les points situés dans chacun des deux demi-espaces sont stockés dans les branches gauche et droite du nœud courant. Par exemple, si un nœud donné divise l'espace selon un plan normal à la direction (Ox), tous les points de coordonnée x inférieure à la coordonnée du point associé au nœud seront stockés dans la branche gauche du nœud. De manière similaire, les points de coordonnée x supérieure à celle du point considéré seront stockés dans la branche droite du nœud.

On peut représenter un nœud l'arbre k-d par la classe (cette classe est une base pour plusieurs variantes) :



Dans un KD-Tree et à un niveau donné, l'arbre prend des décisions de branchement en fonction d'une clé de recherche particulière associée avec ce niveau, appelé le *discriminateur*. En principe, les KD-Tree peuvent être utilisés pour rechercher une clé à travers n'importe quel ensemble de clés arbitraires telles que le nom ou le code postal. Mais en pratique, il est presque toujours utilisé pour effectuer la recherche de coordonnées multidimensionnelles, tels que les emplacements dans l'espace 2D ou 3D. Nous définissons le discriminateur au niveau i en étant *i mod k* pour k dimensions. Par exemple, supposons que nous stockons les données organisées par coordonnées xy. Dans ce cas, k est 2 (il y a deux coordonnées), avec le

x champ de coordonnées arbitrairement désigné par la clé 0, et le champ de coordonnées y associé à la clé 1. A chaque niveau, le discriminateur alterne entre x et y. Ainsi, un noeud N au niveau 0 (la racine) aurait dans son sous-arbre gauche seulement les nœuds dont les valeurs x sont plus petite que Nx (car x est la clé de recherche 0 et 0 mod 2 = 0). Le sous-arbre droit contient les nœuds dont les valeurs x sont supérieures à Nx. Un nœud M au niveau 1 n'aurait dans son sous-arbre gauche que les nœuds dont les valeurs y sont inférieures à My. Il n'y a pas de restriction sur les valeurs relatives de Mx et les valeurs x des descendants de M, car les décisions de branchement faites en M sont basées uniquement sur la coordonnée y.



Dans l'exemple ci-dessus si nous cherchons P = (69,50), nous le comparons d'abord avec le point stocké à la racine (A dans la figure). Si P correspond à l'emplacement de A, la recherche est terminée.

Dans cet exemple, les positions ne correspondent pas, la recherche doit donc continuer. La valeur x de A est comparé à celle de P pour déterminer dans quelle direction brancher. Comme La valeur de Ax (40) est inférieure à la valeur x de P (69), on se branche sur le sous-arbre de droite (toutes les villes avec une valeur x supérieure ou égale à 40 sont dans le sous-arbre de droite).

Ay n'affecte pas la décision sur la manière de se brancher à ce niveau. Au deuxième niveau, P ne correspond pas à la position de l'enregistrement C, donc une autre branche doit être prise. Cependant, à ce niveau, nous dérivons en fonction des valeurs y relatives du point P et de l'enregistrement C (car 1 mod 2 = 1, ce qui correspond à y-coordonnée). Parce que la valeur de Cy de 10 est inférieure à la valeur de Py de 50, nous branche vers la droite. À ce stade, P est comparé à

la position de D. Une correspondance est établie et la recherche est réussie.

Il vous est demandé de coder la structure de données KD-tree avec un algorithme de recherche du plus proche voisin.

En plus de l'algorithme de recherche du plus proche voisin, votre structure devra être capable de créer un KD-Tree, ajouter des nœuds, supprimer des nœuds, rechercher des nœuds.

Exercice 2 : Hypergraph et référencement web

L'indexation du web est au cœur de la navigation sur le World Wide Web. Il s'agit des différentes méthodes qui permettent de parcourir et de classifier la quasi-totalité des pages web. Les moteurs de recherche utilisent des algorithmes représentant le web sous forme d'un graphe afin d'estimer la réputation de la qualité de chaque page. Chaque page est représentée par un nœud, et les liens de redirection d'une page à une autre sont représentés par un arc. Les pages ayant la meilleure réputation apparaissent alors en premier dans les résultats d'une recherche d'un utilisateur si elle correspond avec les termes de sa recherche. C'est donc un élément critique pour nombre d'acteurs du web car beaucoup d'entreprises dépendent de leur classement dans les résultats des différents moteurs de recherche.

De nombreuses stratégies d'analyse de liens ont été réalisées depuis les années 90, nous allons nous concentrer sur deux d'entre elles :

- Indegree
- PageRank qui est utilisé par le moteur de recherche de Google.

Ces deux stratégies consistent à donner un score à chaque page web. Si une page possède un score élevé, cela signifie qu'un nombre important de pages possèdent un lien vers celle-ci. On peut donc supposer qu'elle a une meilleure visibilité et donc une meilleure réputation en termes de qualité.

Une solution afin d'améliorer le classement des pages web consiste à représenter le web sous la forme d'un hypergraphe orienté plutôt qu'un graphe. Cela permet de limiter la redondance des connexions, et donc d'améliorer le classement obtenu.

A partir du graphe de base, les nœuds (des pages web), sont regroupés en blocs non séquents. Les arêtes (les liens) sont utilisées pour construire les hyperarcs. Un hyperarc relie un bloc B à une page p si et seulement si au moins une des pages du bloc pointe vers p.

On ne prend pas en compte les liens d'une page vers elle-même dans le processus.

L'intérêt de regrouper les pages par blocs est de ne représenter que les connexions indépendantes. En effet, avec un graphe simple, si toutes les pages d'un bloc pointent vers une même autre page, on a alors autant d'arêtes que de pages dans le bloc, alors que ces pages peuvent par exemple avoir été créés par la même personne.

Le résultat de la représentation obtenue avec un hypergraphe dépend de la granularité choisie pour les blocs : plus celle-ci est fine (un bloc = une page), plus une page a de l'influence. En jouant sur la granularité, on peut ainsi limiter l'influence des pages dépendantes, rendre les liens plus pertinents. Il est nécessaire de choisir cette granularité selon l'objectif. En effet, plus celle-ci est élevée, plus le nombre d'hyperarcs est faible. Une granularité trop grande entraînera donc un nombre trop petit, et inversement, ce qui peut fausser la réputation des pages. On considère ici trois niveaux de granularité, basés sur l'URL des pages (faible coût de calcul) :

- Regroupement par page : un bloc est composé d'une seule page web. Cela revient à utiliser le modèle de base.
- Regroupement par domaine : un bloc est composé de toutes les pages appartenant à un même domaine.
- Regroupement par hôte : un bloc est composé de toutes les pages d'un même hôte.

Il vous est demandé de mettre en place et d'implémenter les deux algorithmes cités plus hauts en utilisant les hypergraphes. 2 datasets pour vos tests sont disponibles.

Exercice 3 : Labyrinthes

Pour cet exercice, le labyrinthe généré sera un labyrinthe parfait, ce qui implique quelques caractéristiques :

- Tout point à l'intérieur du labyrinthe est accessible depuis n'importe quel autre point de celui-ci.
- L'absence de boucle : il n'existe qu'un seul chemin pour accéder à chaque point du labyrinthe.

L'algorithme du **Growing Tree** permet de générer un labyrinthe parfait. L'idée est de partir d'une grille composée d'éléments inexplorés et d'un élément vide, le point d'entrée. Nous allons prendre aléatoirement un point inexploré adjacent à un point vide, regarder s'il est possible de creuser ce point pour en faire un nouveau point vide, sinon en faire un mur. Il va donc s'agir de répéter cette étape, en prenant soin de ne faire qu'un seul point de sortie.

Il vous est demandé d'écrire un programme qui permet de :

- Générer un labyrinthe de taille donnée avec l'algorithme **Growing Tree**.
- Après avoir placé arbitrairement une source et une destination, résoudre ce labyrinthe en utilisant :
    - BFS
    - A*

Exercice 4 : Compression d'images

On s'intéresse ici à la représentation des images par des quadtrees, utilisés dans l'analyse, la compression et la synthèse d'images. Une image est dite simple si elle est de couleur uniforme. Étant donné un medium graphique (écran, imprimante, etc.) carré, qui mesure 1024 × 1024 pixels, on peut représenter une image par une structure d'arbre, appelé quadtree, dans laquelle chaque nœud a exactement zéro ou quatre fils comme celle vue en classe. L'idée de base est la suivante : si une image associée à un nœud n'est pas simple, on la découpe en quatre morceaux de même taille et les fils du nœud correspondant sont les quadtrees associés aux quatre sous-images. Si au contraire l'image est simple, alors elle est caractérisée par une information unique, sa couleur ; dans ce cas, le nœud porte cette information et n'a pas de fils. D'où le principe de compression, nous n'avons pas besoin de stocker l'information pixel par pixel.

En utilisant les fichiers sources fournis ainsi que les exemples d'images, créer un programme qui permet de :

- Encoder une image en quadtree
- Décoder un quadtree en image